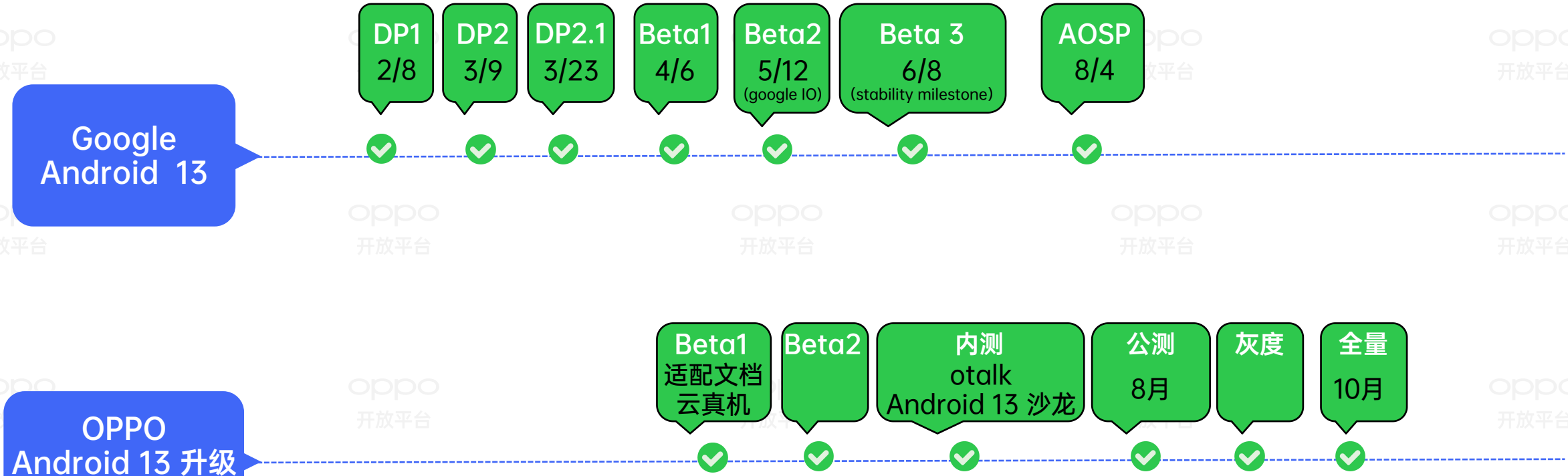
2022

•Android 13
•三方兼容风险项

目录

CATALOGUE

适配计划同步


Google
Android 13
DP1
2/8
DP2
3/9
DP2.1
3/23
Beta1
4/6
Beta2
5/12
(google IO)
Beta 3
6/8
(stability milestone)
AOSP
8/4
OPPO
Android 13 升级
(FindX5 pro等)
Beta1
适配文档
云真机
Beta2
内测
otalk
Android 13 沙龙
公测
8月
灰度
全量
10月

新特性讲解

安全和隐私

- 发通知权限
- 读取媒体文件权限
- 附近WIFI设备权限
- 更安全地导出上下文注册的接收器
- 部分通讯接口权限降级
- 剪贴板擦除

……

系统优化

- 前台服务 (FGS) 任务管理器
- 电池资源利用率

……

新功能和 API

- 照片选择器和API
- OpenJDK 11 updates
- Programmable shaders
- 快捷设置 API
- ActivityEmbedding

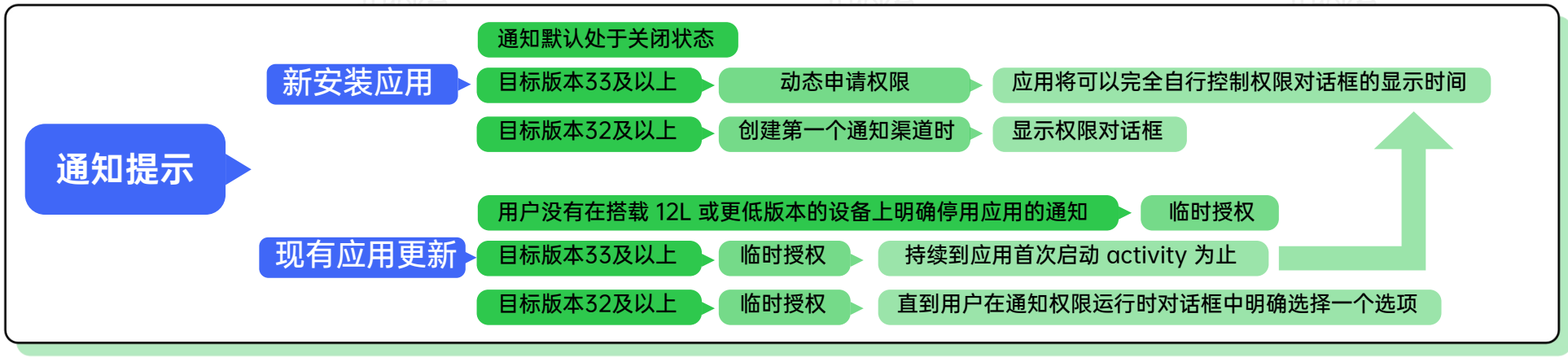
……

发送通知权限（POST_NOTIFICATIONS）

●**谷歌意图：**加强通知的管控，打扰必询问。Android 13 中的通知访问会根据正在运行的应用程序的目标API级别进行不同的处理。

●**变更内容：**

通知提示显示规则

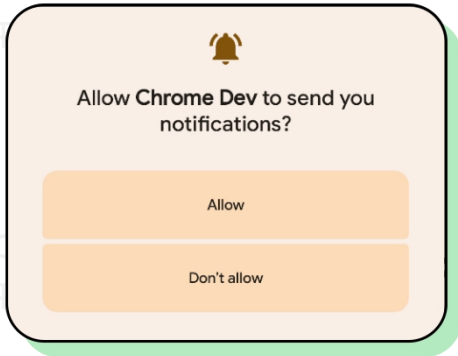


提示框操作规则

允许　可以发送通知；

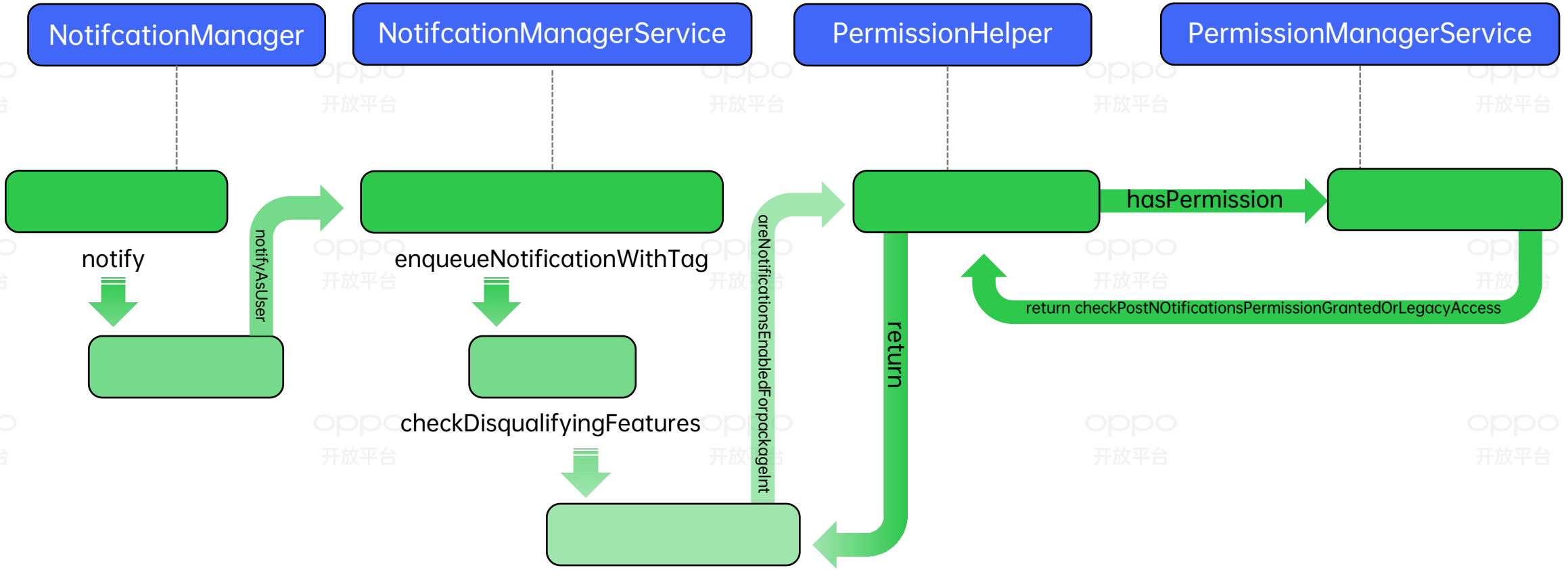
不允许　除去豁免的，无法发送通知；（注意：如果用户点按不允许，即使仅点按一次，系统也不会再次提示用户，除非他们卸载并重新安装您的应用。）

滑开对话框　有临时授权，可以持续保留；没有临时授权，不能发



发送通知权限（POST_NOTIFICATIONS）

鉴权流程：



1、在NotificationManagerService#checkDisqualifyingFeatures发起鉴权
2、在PermissionManagerService#checkPostNotificationsPermissionGrantedOrLegacyAcces获取权限结果

发送通知权限（POST_NOTIFICATIONS）

关键函数：

```
boolean checkDisqualifyingFeatures(int userId, int uid, int id, String
tag,
            NotificationRecord r, boolean isAutogroup) {
        Notification n = r.getNotification();
......
        boolean isBlocked = !areNotificationsEnabledForPackageInt(pkg,
uid);
        synchronized (mNotificationLock) {
            isBlocked |= isRecordBlockedLocked(r);
        }
        if (isBlocked && !(n.isMediaNotification() || isCallNotificati
on(pkg, uid, n))) {
            if (DBG) {
                Slog.e(TAG, "Suppressing notification from package " +
r.getSbn().getPackageName()
                        + " by user request.");
            }
            mUsageStats.registerBlocked(r);
            return false;
        }

        return true;
    }
```

NotificationManagerService
#checkDisqualifyingFeatures函数

```
@Override
public int checkPostNotificationsPermissionGrantedOrLegacyAccess(int uid
  int granted = PermissionManagerService.this.checkUidPermission(uid,
                  POST_NOTIFICATIONS);
  AndroidPackage pkg = mPackageManagerInt.getPackage(uid);
  if (pkg == null) {
      Slog.e(LOG_TAG, "No package for uid " + uid);
      return granted;
  }
  if (granted != PackageManager.PERMISSION_GRANTED
          && pkg.getTargetSdkVersion() >= Build.VERSION_CODES.M) {
      int flags = PermissionManagerService.this.getPermissionFlags(pkg.g
            POST_NOTIFICATIONS, UserHandle.getUserId(uid));
      if ((flags & PackageManager.FLAG_PERMISSION_REVIEW_REQUIRED) != 0)
        return PackageManager.PERMISSION_GRANTED;
       }
   }
   return granted;
}
```

PermissionManagerService#checkPostNotifications
PermissionGrantedOrLegacyAcces函数

发送通知权限（POST_NOTIFICATIONS）

关键函数：

新安装应用检查权限ParsingPackageUtils#convertCompatPermissions

```
private static void convertCompatPermissions(ParsingPackage pkg) {
    for (int i = 0, size = CompatibilityPermissionInfo.COMPAT_PERMS.length; i < size; i++) {
        final CompatibilityPermissionInfo info = CompatibilityPermissionInfo.COMPAT_PERMS[i];
        if (pkg.getTargetSdkVersion() >= info.getSdkVersion()) {
            break;
        }
        if (!pkg.getRequestedPermissions().contains(info.getName())) {
            pkg.addImplicitPermission(info.getName());
        }
    }
}
```

```
public static final CompatibilityPermissionInfo[] COMPAT_PERMS =
        new CompatibilityPermissionInfo[]{
            new CompatibilityPermissionInfo(Manifest.permission.POST_NOTIFICATIONS,
                android.os.Build.VERSION_CODES.TIRAMISU),
            new CompatibilityPermissionInfo(Manifest.permission.WRITE_EXTERNAL_STORAGE,
                android.os.Build.VERSION_CODES.DONUT),
            new CompatibilityPermissionInfo(Manifest.permission.READ_PHONE_STATE,
                android.os.Build.VERSION_CODES.DONUT)
        };
```

应用创建通知渠道时由系统强制弹框请求通知权限

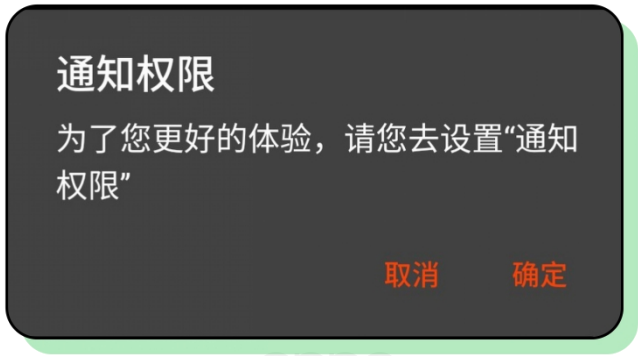
```
private void createNotificationChannelsImpl(String pkg, int uid,
                                            ParceledListSlice channelsList, int startingTaskId) {
                    List<NotificationChannel> channels = channelsList.getList();
                    final int channelsSize = channels.size();
                    。。。。。。
                    for (int i = 0; i < channelsSize; i++) {
                                        。。。。。。
                                        if (needsPolicyFileChange) {
                                                                                                                    }
                                                            。。。。。。
                                                            if (!hadChannel && hasChannel && !hasRequestedNotificationPermission
                                                                                                    && startingTaskId != ActivityTaskManager.INVALID_TASK_ID) {
                                                                                。。。。。。
                                                                                mHandler.post(new ShowNotificationPermissionPromptRunnable(pkg,
                                                                                                                        UserHandle.getUserId(uid), startingTaskId,
                                                                                                                        mPermissionPolicyInternal));
                                                            }
                                        }
                    }
                    。。。。。。
}
```

PS：弹框控制逻辑在shouldForceShowNotificationPermissionRequest

发送通知权限（POST_NOTIFICATIONS）

主要应用场景影响：

- 应用如果本身有自己的弹框提示引导，在产品设计上出现冲突，如图某应用弹出的引导用户通知权限设置。
- 用户在不允许应用发通知的后，存在感觉来消息没有及时收到的用户体验；
- 升级用户体验的一致性；



开发者适配须知：

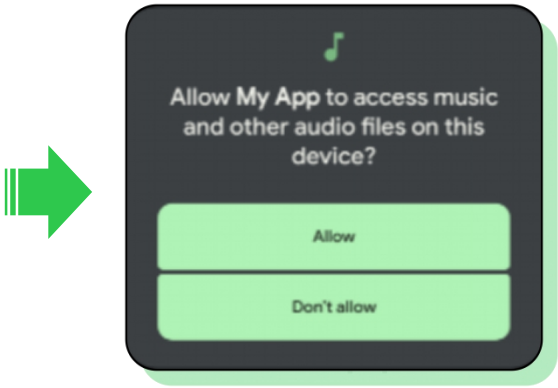
- 产品交互上避免冲突；
- 使用引导描述界面，让用户同意；

读取媒体文件权限

谷歌意图：对于目标版本为 Android 13，细化READ_EXTERNAL_STORAGE权限

变更内容：

- 将 READ_EXTERNAL_STORAGE 权限细化为3个权限：
  - READ_MEDIA_IMAGES
  - READ_MEDIA_VIDEO
  - READ_MEDIA_AUDIO



- 应用先前已被授予 READ_EXTERNAL_STORAGE 权限 就无需再次请求这些读取媒体文件权限;
- 目标平台为 Android 13 的应用如需访问由其他应用创建的媒体文件，必须满足以下条件之一：
  - 应用已获得 READ_MEDIA_IMAGES 权限可访问 MediaStore.Images 媒体集合中的文件
  - 应用已获得 READ_MEDIA_VIDEO 权限可访问 MediaStore.Video 媒体集合中的文件
  - 应用已获得 READ_MEDIA_AUDIO 权限可访问 MediaStore.Audio 媒体集合中的文件

ps:如果您同时请求 READ_MEDIA_IMAGES 权限和 READ_MEDIA_VIDEO 权限，则只会出现一个系统权限对话框。

读取媒体文件权限

关键函数：

```
public static boolean checkPermissionReadImages(
        @NonNull Context context,
        int pid,
        int uid,
        @NonNull String packageName,
        @Nullable String attributionTag,
        boolean targetSdkIsAtLeastT) {
    String permission = targetSdkIsAtLeastT ? READ_MEDIA_IMAGES : READ_EXTERNAL_STORAGE;

    if (!checkPermissionForPreflight(context, permission, pid, uid, packageName)) {
        return false;
    }

    return checkAppOpAllowingLegacy(context, OPSTR_READ_MEDIA_IMAGES, pid,
            uid, packageName, attributionTag,
            generateAppOpMessage(packageName, sOpDescription.get()));
}
```

READ_MEDIA_IMAGES

```
public static boolean checkPermissionReadVideo(
        @NonNull Context context,
        int pid,
        int uid,
        @NonNull String packageName,
        @Nullable String attributionTag,
        boolean targetSdkIsAtLeastT) {
    String permission = targetSdkIsAtLeastT ? READ_MEDIA_VIDEO : READ_EXTERNAL_STORAGE;

    if (!checkPermissionForPreflight(context, permission, pid, uid, packageName)) {
        return false;
    }

    return checkAppOpAllowingLegacy(context, OPSTR_READ_MEDIA_VIDEO, pid,
            uid, packageName, attributionTag,
            generateAppOpMessage(packageName, sOpDescription.get()));
}
```

READ_MEDIA_VIDEO

```
public static boolean checkPermissionReadAudio(
        @NonNull Context context,
        int pid,
        int uid,
        @NonNull String packageName,
        @Nullable String attributionTag,
        boolean targetSdkIsAtLeastT) {

    String permission = targetSdkIsAtLeastT ? READ_MEDIA_AUDIO : READ_EXTERNAL_STORAGE;

    if (!checkPermissionForPreflight(context, permission, pid, uid, packageName)) {
        return false;
    }
    return checkAppOpAllowingLegacy(context, OPSTR_READ_MEDIA_AUDIO, pid,
            uid, packageName, attributionTag,
            generateAppOpMessage(packageName, sOpDescription.get()));
}
```

READ_MEDIA_AUDIO

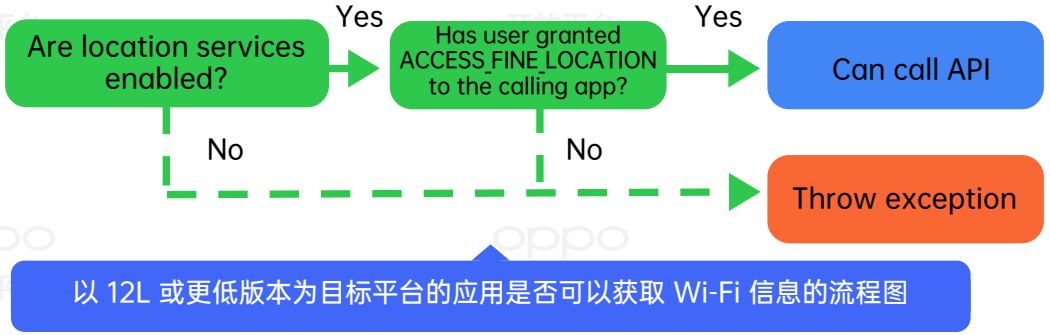
开发者适配需知：

保持与旧版本Android的兼容性,READ_EXTERNAL_STORAGE的maxSdkVersion设为32。

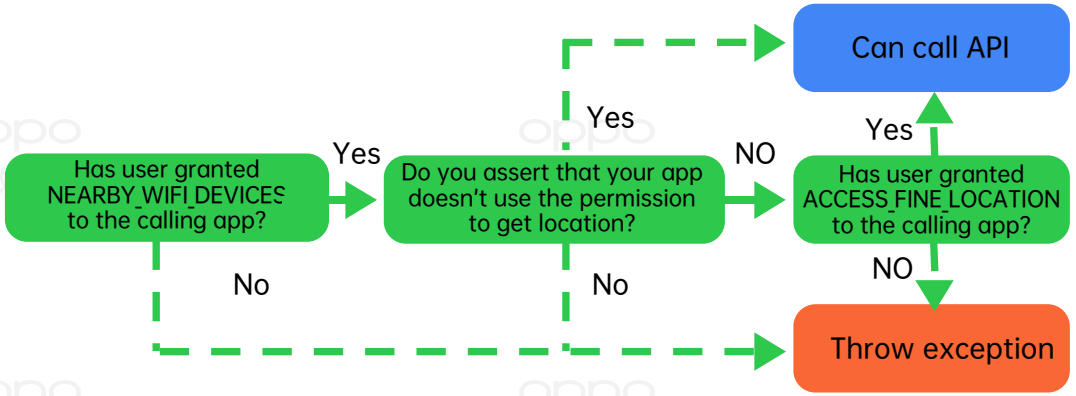
```
<manifest ...>
    <!-- Required only if your app targets Android 13. -->
    <!-- Declare one or more the following permissions only if your app needs
    to access data that's protected by them. -->
    <uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
    <uses-permission android:name="android.permission.READ_MEDIA_AUDIO" />
    <uses-permission android:name="android.permission.READ_MEDIA_VIDEO" />

    <!-- Required to maintain app compatibility. -->
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
                     android:maxSdkVersion="32" />
    <application ...>
        ...
    </application>
</manifest>
```

附近的WIFI设备权限

由于可以通过跟踪附近的Wi-Fi AP和蓝牙设备来推断设备的位置，谷歌决定禁止应用程序访问蓝牙或Wi-Fi扫描结果，除非这些应用程序拥有位置权限。在 Android 13 中，谷歌同样将Wi-Fi扫描与位置权限分离(Android S ,把BT和位置分离)。Android 13 为管理设备与周围 Wi-Fi 热点连接的应用添加 NEARBY_WIFI_DEVICES 运行时权限(属于 NEARBY_DEVICES 权限组)。



以 12L 或更低版本为目标平台的应用是否可以获取 Wi-Fi 信息的流程图



以 Android 13 或更高版本为目标平台的应用是否可以获取 Wi-Fi 信息的流程图

适配需知：

向下兼容:由于 NEARBY_WIFI_DEVICES 权限仅适用于 Android 13 或更高版本，应保留对 ACCESS_FINE_LOCATION 的所有声明，以便在您的应用中提供向下兼容性。如果您的应用不会使用 Wi-Fi API 推导物理位置信息，就可以将此权限的最高 SDK 版本设为 32

```
<manifest ...>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
                     android:maxSdkVersion="32" />
    <application ...>
        ...
    </application>
</manifest>
```

以 Android 13 为目标平台时，如果应用不会通过 Wi-Fi API 推导物理位置，请在清单文件中将 usesPermissionFlags 属性设为 neverForLocation。

```
<manifest ...>
    <uses-permission android:name="android.permission.NEARBY_WIFI_DEVICES"
                     android:usesPermissionFlags="neverForLocation" />
    <application ...>
        ...
    </application>
</manifest>
```

附近的WIFI设备权限

变更内容：

- 需要新权限（NEARBY_WIFI_DEVICES）的 API：

  WifiManager：startLocalOnlyHotspot()

  WifiAwareManager：attach()

  WifiAwareSession：publish()、subscribe()

  WifiP2pManager：addLocalService()、connect()、createGroup()、discoverPeers()、discoverServices()、requestDeviceInfo()、requestGroupInfo()、requestPeers()

  WifiRttManager：startRanging()

startLocalOnlyHotspot接口修改前后对比

```
@RequiresPermission(allOf = {
        android.Manifest.permission.CHANGE_WIFI_STATE,
        android.Manifest.permission.ACCESS_FINE_LOCATION})
public void startLocalOnlyHotspot(LocalOnlyHotspotCallback callback,
        @Nullable Handler handler) {
    Executor executor = handler == null ? null : new HandlerExecutor(handler);
    startLocalOnlyHotspotInternal(null, executor, callback);
}
```

```
@RequiresPermission(allOf = {CHANGE_WIFI_STATE, NEARBY_WIFI_DEVICES}, conditional = true)
public void startLocalOnlyHotspot(LocalOnlyHotspotCallback callback,
        @Nullable Handler handler) {
    Executor executor = handler == null ? null : new HandlerExecutor(handler);
    startLocalOnlyHotspotInternal(null, executor, callback);
}
```

- 仍需要位置信息权限（ACCESS_FINE_LOCATION ）的API：

  WifiManager：getScanResults()、startScan()

更安全地导出上下文注册的接收器

- 目标版本为 Android 13 及以上应用；
- 必须为每个广播接收器指定 RECEIVER_EXPORTED 或 RECEIVER_NOT_EXPORTED。否则，当您尝试注册广播接收器时，系统会抛出 SecurityException；
- 此变更为 Android S更安全的组件输出的延续；

2.1、更安全的组件输出

1）背景
以Android 12为目标平台的应用（target API 级别31），如果包含用 intent filters 修饰的activities，services，broadcast receivers，则必须为这些应用组件显式声明android:exported属性，表明是否支持其它应用调用当前组件。
谷歌官网特性介绍：
https://developer.android.com/about/versions/12/behavior-changes-12#exported
2）兼容性影响
如果您的应用包含使用intent filter但未声明android:exported的Activity，Service，broadcast receiver，开发过程中会提示以下警告消息，具体取决于您使用的Android Studio版本：
Android Studio 2020.3.1 Canary 11或更高版本
清单文件中将出现以下警告：
When using intent filters, please specify android:exported as well
当您尝试编译应用程序时，会生成以下错误消息：
Manifest merger failed : Apps targeting Android 12 and higher are required
to specify an explicit value for android:exported when the corresponding \
component has an intent filter defined.
旧版Android Studio
如果您尝试安装该应用程序，Logcat将显示以下错误消息：
Installation did not succeed.
The application could not be installed: INSTALL_FAILED_VERIFICATION_FAILURE
List of apks:
[0] '.../build/outputs/apk/debug/app-debug.apk'
Installation failed due to: 'null'

Android S 更安全的组件输出

```
// This broadcast receiver should be able to receive broadcasts from other apps.
// This option causes the same behavior as setting the broadcast receiver's
// "exported" attribute to true in your app's manifest.
context.registerReceiver(sharedBroadcastReceiver, intentFilter,
    RECEIVER_EXPORTED);

// For app safety reasons, this private broadcast receiver should **NOT**
// be able to receive broadcasts from other apps.
context.registerReceiver(privateBroadcastReceiver, intentFilter,
    RECEIVER_NOT_EXPORTED);
```

Android 13 更安全地导出上下文注册的接收器

开发者适配须知：请注意应用引用的jar包的整改

其他权限

谷歌意图：Android 13 增加了剪贴板自动清除功能。在经过设定的时间后，将自动从全局剪贴板中清除主剪辑。

变更内容：

- 每次执行复制/读取（写入剪贴板setPrimaryClip，图1；读getPrimaryClip，图2）时，会重置一个消息，timeout（60min）后，自动清除剪贴板内存中的内容。即60min内，如果一直没有写入剪贴板的操作，剪贴板的内容会被自动清除，此时粘贴功能将失效。

图1

```
@Override
public void setPrimaryClip(ClipData clip, String callingPackage, @UserIdInt int userId) {
    checkAndSetPrimaryClip(clip, callingPackage, userId, callingPackage);
}

@Override
public void setPrimaryClipAsPackage(
        ClipData clip, String callingPackage, @UserIdInt int userId, String sourcePackage) {
    getContext().enforceCallingOrSelfPermission(Manifest.permission.SET_CLIP_SOURCE,
            "Requires SET_CLIP_SOURCE permission");
    checkAndSetPrimaryClip(clip, callingPackage, userId, sourcePackage);
}

private void checkAndSetPrimaryClip(
        ClipData clip, String callingPackage, @UserIdInt int userId, String sourcePackage) {
    if (clip == null || clip.getItemCount() <= 0) {
        throw new IllegalArgumentException("No items");
    }
    final int intendingUid = getIntendingUid(callingPackage, userId);
    final int intendingUserId = UserHandle.getUserId(intendingUid);
    if (!clipboardAccessAllowed(AppOpsManager.OP_WRITE_CLIPBOARD, callingPackage,
            intendingUid, intendingUserId)) {
        return;
    }
    checkDataOwner(clip, intendingUid);
    synchronized (mLock) {
        scheduleAutoClear(userId);
        setPrimaryClipInternalLocked(clip, intendingUid, sourcePackage);
    }
}

private void scheduleAutoClear(@UserIdInt int userId) {
    final long oldIdentity = Binder.clearCallingIdentity();
    try {
        if (DeviceConfig.getBoolean(DeviceConfig.NAMESPACE_CLIPBOARD,
                PROPERTY_AUTO_CLEAR_ENABLED, true)) {
            mClipboardClearHandler.removeEqualMessages(ClipboardClearHandler.MSG_CLEAR,
                    userId);
            Message clearMessage = Message.obtain(mClipboardClearHandler,
                    ClipboardClearHandler.MSG_CLEAR, userId, 0, userId);
            mClipboardClearHandler.sendMessageDelayed(clearMessage,
                    getTimeoutForAutoClear());
        }
    } finally {
        Binder.restoreCallingIdentity(oldIdentity);
    }
}
```

图2

```
@Override
public ClipData getPrimaryClip(String pkg, @UserIdInt int userId) {
    final int intendingUid = getIntendingUid(pkg, userId);
    final int intendingUserId = UserHandle.getUserId(intendingUid);
    if (!clipboardAccessAllowed(AppOpsManager.OP_READ_CLIPBOARD, pkg,
            intendingUid, intendingUserId)
            || isDeviceLocked(intendingUserId)) {
        return null;
    }
    synchronized (mLock) {
        try {
            addActiveOwnerLocked(intendingUid, pkg);
        } catch (SecurityException e) {
            // Permission could not be granted - URI may be invalid
            Slog.i(TAG, "Could not grant permission to primary clip. Clearing clipboard.");
            setPrimaryClipInternalLocked(null, intendingUid, pkg);
            return null;
        }

        PerUserClipboard clipboard = getClipboardLocked(intendingUserId);
        showAccessNotificationLocked(pkg, intendingUid, intendingUserId, clipboard);
        notifyTextClassifierLocked(clipboard, pkg, intendingUid);
        if (clipboard.primaryClip != null) {
            scheduleAutoClear(userId);
        }
```

其他权限

部分通讯接口权限降级

- 新增权限READ_BASIC_PHONE_STATE，放开一些APK必现READ_PHONE_STATE这个危险权限
- 允许以非危险权限只读访问手机状态，包括网络类型、软件版本等信息。

  getDeviceSoftwareVersion()、getDataNetworkType()、getVoiceNetworkType()、isDataEnabled() ....
- 原READ_PHONE_STATE依然生效

```
/**
 * Returns the software version number for the device, for example,
 * the IMEI/SV for GSM phones. Return null if the software version is
 * not available.
 * <p>
 */
@RequiresPermission(anyOf = {
        android.Manifest.permission.READ_PHONE_STATE,
        android.Manifest.permission.READ_BASIC_PHONE_STATE})
@Nullable
public String getDeviceSoftwareVersion() {
    return getDeviceSoftwareVersion(getSlotIndex());
}
```

限制legacy apps可以下载文件到其他应用的私有目录

谷歌意图：加强通过下载服务将文件下载到其他应用私有目录的限制

变更内容：

- 在 Android 13 上，无法通过下载服务下载到其他应用私有目录
- 在 Android 13 之前，如果是legacyMode的应用，并且具备 WRITE_EXTERNAL_STORAGE权限，是可以下载到其他应用的目录

用场景（用户影响）：

1、应用使用DownloadProvider 不可访问其他应用的external 私有目录，比如/storage/emulated/0/Android/obb|data/XXX/

如果您的应用以 Android 10（API 级别 29）或更低版本为目标平台，您可以暂时在正式版应用中停用分区存储。不过，如果您以 Android 10 为目标平台，则需要在应用的清单文件中将 `requestLegacyExternalStorage` 的值设置为 `true`：

```
<manifest ... >
  <!-- This attribute is "false" by default on apps targeting
       Android 10. -->
  <application android:requestLegacyExternalStorage="true" ... >
    ...
  </application>
</manifest>
```

前台服务 (FGS) 任务管理器

谷歌意图：Android 13 的新前台服务（FGS）任务管理器显示当前运行前台服务的应用程序列表。每个应用程序旁边都会有一个"停止"按钮。

变更内容：

- 与从最近用过屏幕"向上滑动"和"强行停止"对比；

	FGS任务管理器	向上滑动	强行停止
从历史记录中移除	✓	✓	✓
立即从内存中移除	✓		✓
停止媒体播放	✓		✓
停止 FGS/移除关联的通知	✓		✓
移除 activity 返回堆栈		✓	✓
取消Job			✓
取消Alarm			✓

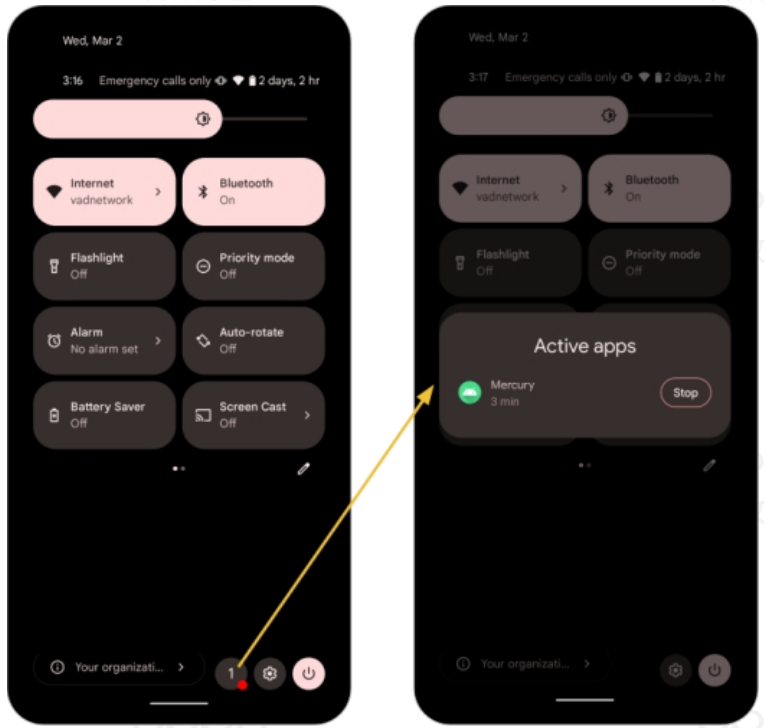


图 1. Android 13 的设备上的 FGS 任务管理器工作流程。

出现时机：

- 如果系统检测到您的应用长时间运行某项前台服务（在 24 小时的时间段内至少运行 20 小时），便会发送通知邀请用户与 FGS 任务管理器互动。

```
/**
 * Default value to {@link #mMaxTrackingDuration}.
 */
static final long DEFAULT_BG_FGS_LONG_RUNNING_WINDOW = ONE_DAY;

/**
 * Default value to {@link #mBgFgsLongRunningThresholdMs}.
 */
static final long DEFAULT_BG_FGS_LONG_RUNNING_THRESHOLD = 20 * ONE_HOUR;
```

前台服务 (FGS) 任务管理器

变更内容：

[豁免]

完全不会显示在 FGS 任务管理器中

- 系统级应用
- 安全应用，即具有 ROLE_EMERGENCY 角色的应用
- 处于demo模式的设备上的应用

不能被用户停止的：它们会显示在 FGS 任务管理器中，但应用名称旁边没有可以供用户按的停止按钮；

- 设备所有者应用
- 资料所有者应用
- 常驻应用
- 具有 ROLE_DIALER 角色的应用

设备所有者应用

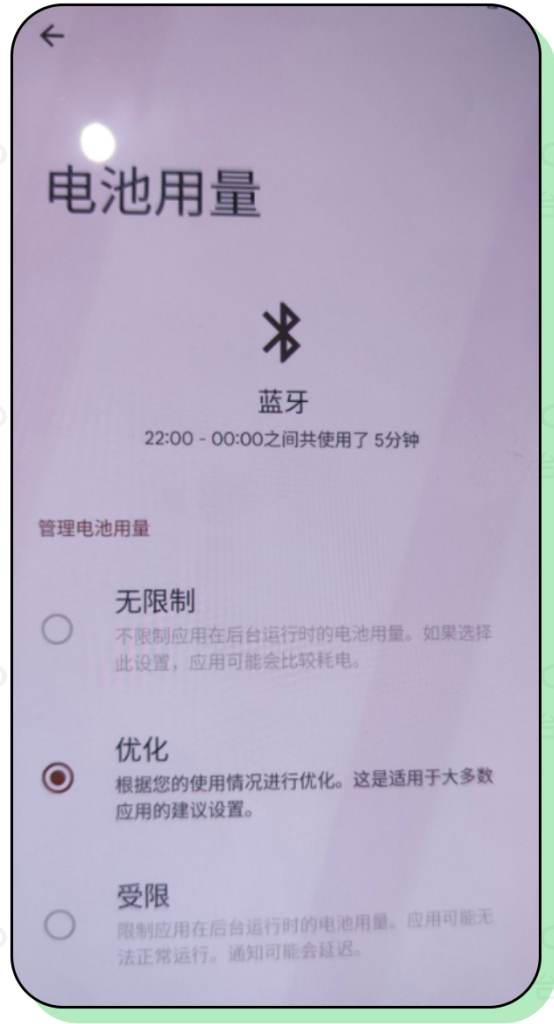
资料所有者应用

电池资源利用率

谷歌意图： Android 13 推出了以下电池保护措施

变更内容：

- 更新了系统何时将你的应用程序放入“受限”应用程序备用存储桶的规则。
  - 用户有 8 天没有与您的应用互动。如果用户与另一个绑定到您应用的服务的应用互动，系统也会将您的应用视为“使用过”。
  - 您的应用在 24 小时内调用了过多的广播或绑定。**AppBindServiceEventsTracker/AppBroadcastEventsTracker**
  - 您的应用在 24 小时内消耗了大量的设备电池电量。**AppBatteryTracker**
- 当用户将应用程序置于后台电池使用的“受限”状态时，应用程序执行的新限制。
  - 无法启动前台服务
  - 现有的前台服务会从前台移除
  - 不会触发 **Alarm**
  - 不会job作业 （以上 **Android 9** 就有的规则）
  - **Android 13** 为目标平台时:除非应用因其他原因启动，否则系统不会传送广播 **BOOT_COMPLETED、LOCKED_BOOT_COMPLETED**
- 新的系统通知，提醒用户后台电池过度使用和长期运行的前台服务。
  - **Android 13** 引入了一个新的系统通知，当您的应用在 **24** 小时内消耗了大量设备电池电量时，就会显示该通知。



电池资源利用率

豁免

存在以下情况的应用不会受到 Android 13 中引入的任何省电措施的影响：

- 系统应用和系统绑定应用
- 配套设备应用
- 处于演示模式的设备上运行的应用
- 设备所有者应用
- 资料所有者应用
- 常驻应用
- VPN 应用
- 具有 ROLE_DIALER 角色的应用
- 用户在系统设置中明确指定提供"无限制"功能的应用

在以下情况下，您的应用可以免于进入"受限"应用待机模式存储分区，并可以绕过"8 天无活动"触发器：

- 具有活跃的 widget
- 至少被授予下列其中一种权限：SCHEDULE_EXACT_ALARM、ACCESS_BACKGROUND_LOCATION

适配支持

OPPO将提供以下支持

兼容性适配指导文档

开发者预览版本

云真机/云测

适配答疑交流社群

新特性检测工具

OPPO开发者社区适配支持专区

云真机

为开发者提供提供统一的测试解决方案，助力开发者提升APP适配效率

开发者痛点

- 安卓手机碎片化严重
- 遇到兼容问题难定位难排查
- 太多机型适配购机成本高
- …… ……

OPPO云测平台

为开发者降低不同机型测试成本，极大提升研发效率，开发者痛点逐一击破

云真机　　自动化测试

自动发现问题

通过自动化测试快速发现应用在OPPO手机上的适配问题

快速定位问题

通过远程真机快速排查定位问题

机型全面

支持全球50+OPPO热门机型

本次提供搭载 Android 13 的OPPO Find X5 Pro云真机

使用数据

10,000+

云真机开发者使用人次

480,000+

云真机适配服务次数

其他

✓ **其他兼容性分析将会放在 OPPO 开放平台**

https://open.oppomobile.com/new/developmentDoc/info?id=11314

https://open.oppomobile.com/new/developmentDoc/info?id=11311

Android 13 应用兼容性适配指导

OPPO Android 13 开发者预览版